Bolt Beranek and Newman Inc.

bbn

12

Report No. 3540

TENEX MSG User Manual

Robert H. Thomas, Paul R. Johnson

April 1977

Submitted to:
Defense Advanced Research Projects Agency

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: BBN Report No. 3540	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): TENEX MSG User Manual.		5. TYPE OF REPORT & PERIOD COVERED: Technical 8/1/76 - 5/30/77
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Robert H. Thomas, Paul R. Johnson		8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0773, ARPA Order-2935
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Massachusetts 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Technical rept. 1 Aug 76 - 30 May 77		12. REPORT DATE: April 1977
		13. NUMBER OF PAGES: 28
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): 32p.		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of thes document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2935.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

National Software Works
interprocess communication
TENEX Operating System
Network operating systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the implementation of MSG, the interprocess communication facility for the National Software Works (NSW) system, for the TENEX operating system. It is intended as a reference for programmers who use MSG and as a guide for those responsible for the operation of systems, such as NSW, which uses MSG.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060 100

LB

BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

BBN Report No. 3540 April 1977

TENEX MSG User Manual

Robert H. Thomas

Paul R. Johnson

This work was supported by the Defense Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract No. N00014-75-C-0773.

Contents

1. Introduction

This document describes from a user's point of view the TENEX implementation of MSG, the interprocess communication facility of the National Software Works (NSW) system. While MSG was developed to satisfy the communication requirements of the NSW system, it is our belief that MSG is a generally useful interhost interprocess communication facility which is applicable outside of the NSW environment. This document is intended as a reference for programmers who use MSG and as a guide for those responsible for operation of systems, such as NSW, which use MSG.

The document should be regarded as a TENEX-specific companion to the MSG design specification ("MSG: The Interprocess Communication Facility for the National Software Works", BBN Report No. 3483, MCA Document No. CADD-7612-2411). The reader is assumed to be familiar with the MSG design specification and, in particular, with the semantics of the various MSG primitives which will not be described here.

The remainder of this document is organized as follows:

Section 2 describes the interface to processes (i.e., TENEX forks) supported by MSG. It can be regarded as an MSG programmers guide for TENEX.

Section 3 describes the interface provided by MSG to human users. It describes how to define and run MSG configurations, and how to use the MSG process monitor and debugger.

The appendix specifies the error and disposition codes returned to user processes by MSG.

2. TENEX MSG Process Interface

2.1 General

2.1.1. Calling MSG Primitives.

JSYS 215 is used to call MSG primitives. AC0 is used to specify the primitive desired (see Section 2.2). If the contents of AC0 are less than or equal to 0, then MSG will not interpret the call as an MSG primitive. Rather, MSG will allow normal execution of JSYS 215.

2.1.2. Primitive Parameters.

All of the primitives which create pending events are called with AC1 containing the address (E) of a parameter block. The general format of the parameter block is:

```
E:   Primitive specific parameter
E+1: Byte pointer for name of source/destination process
E+2: Signal
E+3: Return disposition
E+4: Timeout (in ms.)
E+5 -> E+n: Additional primitive specific parameters
```

If E+4 is zero a default value is used for the timeout; the default is currently very large (10 hours). The value 377777777777 (octal) can be used to request an infinite timeout.

2.1.3. Signals.

TENEX supports 3 kinds of signals:

Unblock: meaning the process relinquishes control when it executes the primitive and remains suspended until the pending event associated with the primitive has occurred at which time it resumes execution.

PSI: meaning that the process resumes execution after MSG creates the pending event associated with the primitive. When the pending event occurs, MSG will generate a PSI on a channel specified by the process.

Null: meaning that the process does not wish MSG to signal it when the pending events occurs. When the event occurs the disposition will be delivered to the process but the process will not be signalled. The process resumes execution after MSG creates the pending event associated with the primitive.

2.1.4. Representation of Signals.

A TENEX signal is represented by 24 bits, right justified in a 36 bit word. The format of the word is:

B0-B11: zero: unused in the signal (but used in event handles which are created from signals - see below)
B12-B17: Signal type: The currently defined signal types are:
  B12-17 off: Null
  B12 on: Unblock
  B13 on: PSI
B18-B35: Signal data:
  For Unblock and Null there is no data and B18-B35 should be zero.
  For PSI the signal data is a PSI channel number.

2.1.5. Event Handles.

When a primitive that creates a pending event is executed, MSG returns to the calling process an "event handle" which the process can use in MSG primitives which manipulate pending events (such as Rescind). The event handle is returned in the parameter block at position E+2 (which is the position of the signal parameter on the call).

Event handles are 36 bit quantities which are derived from signals. The format of an event handle is:

B0-B11: Event identifier.
B12-B35: Signal associated with the event.

2.1.6. Dispositions.

A TENEX disposition is a 36 bit quantity. Zero is always used to indicate the success disposition. Minus one (-1) is always used to indicate that the disposition of the primitive has not yet been determined (i.e., that the signal has not yet occurred). A positive disposition indicates an unsuccessful primitive. The format for positive dispositions is:

B0-B19: Zero
B20-B35: 16 bit code which indicates the reason the primitive was unsuccessful (see Appendix for codes).

2.1.7. Process Names.

A process name is represented by a sequence of 8 bit bytes (See Section 5 of MSG Design Specification Document). The bytes are packed 4 8-bit bytes per 36 bit word (left-justified) and have the format.

Bytes 1,2: Host address (right-justified)
Bytes 3,4: Host incarnation #
Bytes 5,6: Process instance #
Byte 7: Count
Bytes 8->n: String of Count characters which identify a generic class.

For a generically addressed message (i.e., a message sent by the SendGenericMessage primitive) the Host incarnation # and Process instance # must both be the special value "unspecified" (=Ø); the Host address may (but need not) have the value "unspecified".

2.1.8. Success and Failure of MSG calls.

After an MSG primitive is initiated (and completed, for those primitives that do not create pending events), MSG returns control to the calling process either at the location +1 (non-skip return) or +2 (skip return) relative to the location of the primitive call.

The non-skip return is used whenever MSG is unable to successfully execute the primitive. In this case MSG returns a code which indicates the reason the primitive failed in AC2. This failure code has the same format as a disposition (see Section 2.1.6). For a primitive that creates a pending event, MSG will use the non-skip return if it is unable to create the pending event; for example, because the call parameters are not well formed.

For primitives that do not create pending events the skip return is used to indicate that the primitive was successfully executed. In general, for primitives that create pending events the skip return is used to indicate that the pending event was successfully created. The success or failure of the primitive will be supplied to the calling process as the disposition when the signal associated with the pending event is generated by MSG.

2.1.9. The Unblock Signal

When a process specifies an Unblock signal, MSG "plants" a WAIT JSYS followed by a JRST instruction in the process address space and restarts the process at the WAIT instruction. When the associated pending event completes, the process is started at the JRST instruction which causes process execution to resume at the skip return location for the primitive. This implementation of the Unblock signal impacts processes in two ways: a portion of the process address space is usurped by MSG for these instructions; and, routines in a process which respond to PSIs and examine the PC (program counter) value when the PSI occurred (a bad practice in general) should be aware that the PC may point to one of the planted instructions. Presently MSG uses the last 8 locations in the process address space (777770-777777) for planting these instructions. Eventually MSG may be changed to plant the instructions in the parameter block supplied on primitive calls.

2.1.10. Messages and TENEX pages.

The TENEX MSG implementation uses TENEX pages as message buffers. For the initial implementation a message must fit within a single TENEX page. Subsequent implementations may permit multiple page messages. All messages are stored in the buffer page beginning at the first word in the page and are packed left-justified, 4 8-bit bytes per 36 bit word. TENEX pages contain 512 words; therefore, the maximum message supported in the initial implementation is 2048 bytes.

2.1.11. Miscellaneous

a. Single fork MSG processes.

There is a limitation in the current TENEX MSG implementation which requires that only the top fork in an MSG process execute MSG primitives. This limitation may be removed in future implementations.

2.2. TENEX MSG Calls.

2.2.1. Primitives That Create Pending Events.

2.2.1.1. SendSpecificMessage.

SendSpecificMesssage (msgarea, pnam, signal, disp, dt, sphndl)

```
CALL:
     Ø/ 1
     1/ E
```

Parameter Block:

```
     E  / Page Number [part of msgarea spec]
     E+1/ Byte Ptr for name of destination process [pnam]
     E+2/ Signal
     E+3/ Return disposition [disp]
     E+4/ Timeout [dt]
     E+5/ Message byte count [part of msgarea spec]
     E+6/ Special handling [sphndl]
```

The message is assumed to begin at the first word of the page specified by E and to be packed left-justified, 4 8-bit bytes per 36 bit word.

The types of special handling currently defined include:

```
     Ø - No special handling
     Bit 28 on: Generic message
                [not valid for SendSpecificMessage primitive]
     Bit 29 on: Sequenced message
                [not implemented]
     Bit 3Ø on: Stream marked message.
                [not implemented]
```

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Handling given is invalid.
Unable to map up parameter block.
Invalid host address in process name.
Unable to map parameter block down.
Unable to map message page in Message Send.
MSG message too long.
Destination process message queue full.
Destination name/handling - generic/specific mismatch.
Generic name not legal for destination process.

Bad incarnation number on destination process.
Insufficient resources to complete primitive.

2.2.1.2. SendGenericMessage.

SendGenericMessage (msgarea, genadr, signal, disp, dt, qwait)

CALL:
    0/ 2
    1/ E

Parameter Block:

    E / Page Number [part of msgarea spec]
    E+1/ Byte Ptr for name of destination process [genadr]
    E+2/ Signal
    E+3/ Return disposition [disp]
    E+4/ Timeout [dt]
    E+5/ Message byte count [part of msgarea spec]
    E+6/ unused

The byte ptr in E+1 points to a generic address (see Section 2.1.7). The qwait parameter is not supported in the initial implementation.

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Handling given is invalid.
Process name given is invalid.
Unable to map up parameter block.
Invalid host address in process name.
Unable to map parameter block down.
Unable to map message page in Message Send.
MSG message too long.
Destination process unknown.
Destination process message queue full.
Destination name/handling - generic/specific mismatch.
Generic name not legal for destination process.
Bad incarnation number on destination process.
Insufficient resources to complete primitive.
Can't allocate a process for generic message.

2.2.1.3. ReceiveSpecificMessage.

ReceiveSpecificMessage (msgarea, srcnam, signal, disp, dt,
sphndl)

CALL:

```
     0/ 3
     1/ E
```

Parameter Block

```
     E  / Page Number [part of msgarea spec]
     E+1/ Byte Ptr for area to return name of source process
          [srcnam]
     E+2/ Signal
     E+3/ Return disposition [disp]
     E+4/ Timeout [dt]
     E+5/ Message byte count set by MSG
     E+6/ Special handling set by MSG [sphndl]
     E+7/ Size (in bytes) of area for source process name.
     E+8/ Number of bytes of source process name returned.
     E+9/ Byte ptr to first byte beyond end of source process
          name returned.
```

If this primitive completes successfully, after the signal is generated by MSG, the page specified in E will contain a message which begins at the first word of the page and is packed left-justified, 4 8-bit bytes per 36 bit word. The length of the message in bytes is returned in E+5. MSG will return the name of the sending process in the area specified in E+1; the size of the area is specified in E+7. If the name is too large to fit in this area, MSG will truncate the name.

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Unable to map up parameter block.
Unable to map parameter block down.
Insufficient resources to complete primitive.

2.2.1.4. ReceiveGenericMessage.

ReceiveGenericMessage (msgarea, srcnam, signal, disp, dt)

CALL:

```
     0/ 4
     1/ E
```

Parameter Block:

```
     E  / Page Number [part of msgarea spec]
     E+1/ Byte Ptr for area to return name of source process
          [srcnam]
     E+2/ Signal
     E+3/ Return disposition [disp]
     E+4/ Timeout [dt]
     E+5/ Message byte count set by MSG
     E+6/ unused
     E+7/ Size (in bytes) of area for source process name.
     E+8/ Number of bytes of source process name returned.
     E+9/ Byte ptr to first byte beyond end of source process
          name returned.
```

This primitive is used to recieve generically addressed messages. Otherwise, it behaves exactly like the ReceiveSpecificMessage primitive (with the minor exception that no sphndl is associated with a generically address message).

If the contents of E+8 is zero, the message delivered is an "initialization message" delivered to the process by MSG to signal an MSG restart. (See discussion of INITCLASSJOB in Section 3.1.4).

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Unable to map up parameter block.
Unable to map parameter block down.
Insufficient resources to complete primitive.

2.2.1.5. SendAlarm.

SendAlarm (acode, pnam, signal, disp, dt)

CALL:
    0/ 5
    1/ E

Parameter Block:

    E / 16 bit alarm code [acode]
    E+1/ Byte Ptr for name of destination process [pnam]
    E+2/ Signal
    E+3/ Return disposition [disp]
    E+4/ Timeout [dt]

Possible failure dispositions include (see Appendix for codes):

    Signal given is invalid.
    Process name given is invalid.
    Unable to map up parameter block.
    Invalid host address in process name.
    Unable to map parameter block down.
    Destination process unknown.
    Generic name not legal for destination process.
    Bad incarnation number on destination process.
    Insufficient resources to complete primitive.
    Process not accepting alarms now.
    Alarm already queued for process.

2.2.1.6. EnableAlarm.

EnableAlarm (acode, srcnam, signal, disp)

CALL:
    0/ 6
    1/ E

Parameter Block:

```
E  / 16 bit alarm code returned by MSG [acode]
E+1/ Byte Ptr for area to return name of source process
     [srcnam]
E+2/ Signal
E+3/ Return disposition [disp]
E+4/ unused
E+5/ unused
E+6/ unused
E+7/ Size (in bytes) of area for source process name.
E+8/ Number of bytes of source process name returned.
E+9/ Byte ptr to first byte beyond end of source process
     name returned.
```

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Unable to map up parameter block.
Unable to map parameter block down.
Enable Alarm already outstanding.
Insufficient resources to complete primitive.

2.2.1.7. OpenConn.

Openconn (conntype, connid, pnam, signal, disp, dt)

CALL:

```
0/ 7
1/ E
```

Parameter Block:

```
E  / 16 bit connection identifier [connid]
E+1/ Byte Ptr for name of destination process [pnam]
E+2/ Signal
E+3/ Return disposition [disp]
E+4/ Timeout [dt]
E+5/ Connection type [conntype]
E+6/ Return connection designator.
E+7/ Mask for PSI channel to be used to signal broken
     connection or 0 for no signal.
```

Connection type is a 16 bit quantity which is right-justified in E+5:

```
B20: on - Binary Pair;
     B30-B35: Connection Size
B21: on - Binary Send:
```

```
        B30-B35: Connection Size
    B22.ne 2 : on - Binary Receive:
        B30-B35: Connection Size
    B23: on - Server TELNET
    B24: on - User TELNET
```

The connection designator returned in E+6 depends upon the connection type:

Binary Send/Binary Receive:	E+6/	JFN
Binary Pair/User TELNET:	E+6/	XWD SendJFN,ReceiveJFN
Server TELNET:	E+6/	TTY Designator.

If the connection is broken after it has been successfully opened but before a CloseConn has been executed, MSG will signal the user process via a PSI on the channel specified by the bit mask in E+7.

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Process name given is invalid.
Unable to map up parameter block.
Invalid host address in process name.
Unable to map parameter block down.
Already have connection of that ID.
Remote site refused connection.
Destination process unknown.
Generic name not legal for destination process.
Bad incarnation number on destination process.
Insufficient resources to complete primitive.
Invalid connection type.
Connection type mis-match.

2.2.1.8. CloseConn.

Closeconn (connid, pnam, signal, disp, dt)

CALL:

```
    0/ 8. [=10 Octal]
    1/ E
```

Parameter Block:

```
    E  / 16 bit connection identifier [connid]
    E+1/ Byte Ptr for name of destination process [pnam]
    E+2/ Signal
    E+3/ Return disposition [disp]
    E+4/ Timeout [dt]
```

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Process name given is invalid.
Unable to map up parameter block.
Invalid host address in process name.
Unable to map parameter block down.
Unknown connection.
Connection not to process named.
Destination process unknown.
Generic name not legal for destination process.
Bad incarnation number on destination process.
Insufficient resources to complete primitive.
User process closed connection.
Received a CLS for connection.
Referenced connection transaction does not exist.
Connection aborted.

2.2.1.9. TerminationSignal.

TerminationSignal (tsignal, disp)

CALL:
0/ 9. [=11 Octal]
1/ E

Parameter Block

E / unused
E+1/ unused
E+2/ Signal
E+3/ Return disposition [disp]

Possible failure dispositions include (see Appendix for codes):

Signal given is invalid.
Unable to map up parameter block.
Unable to map parameter block down.
Insufficient resources to complete primitive.

2.2.2. Primitives That Do Not Create Pending Events.

2.2.2.1. Stopme.

Stopme ()

```
CALL:
     0/ 101. [=145 Octal]
```

2.2.2.2. Rescind.

Rescind (rsignal)

```
CALL:
     0/ 102. [=146 Octal]
     1/ Event handle (see Section 5) [rsignal]
```

Possible failure dispositions include (see Appendix for codes):

Invalid Event Handle in Rescind primitive.
Unable to Rescind.

2.2.2.3. AcceptAlarm.

AcceptAlarms (qaccept)

```
CALL:
     0/ 103. [=147 Octal]
     1/ -1 = true; 0 = false [qaccept]
```

Note: If qaccept is false, any queued, but undelivered, alarm will remained queued.

Possible failure dispositions include (see Appendix for codes):

Alarm Accept code not 0 or -1.

2.2.2.4. ReSynch.

Resynch (pnam)

CALL:
    0/ 104. [=150 Octal]
    1/ Byte ptr to process name [pnam]

This primitive is not supported in the current TENEX implementation.

2.2.2.5. WhoAmI.

WhoAmI

CALL:
    0/ 105. [=151 Octal]
    1/ E

Parameter Block

E / Byte ptr for area to return generic name string for process.
E+1/ Byte Ptr for area to return name of process
E+2/ Host address component of process name [returned right justified]
E+3/ Host incarnation component of process name [returned right justified]
E+4/ Instance number component of process name [returned right justified]
E+5/ Size (in bytes) of area for generic name string.
E+6/ Number of bytes of generic name returned.
E+7/ Size (in bytes) of area for process name.
E+8/ Number of bytes of process name returned.

This primitive may be used by a process to discover its MSG process name. The host address, host incarnation, and process instance number components are always returned. The generic name string and/or the full process name are returned only when E+5 and/or E+7 are non-zero.

Possible failure dispositions include (see Appendix for codes):

Unable to map up parameter block.
Unable to map parameter block down.

3. MSG User Interface.

3.1. General.

A TENEX MSG configuration is implemented as a collection of TENEX user jobs. ALL TENEX processes which communicate via MSG must execute under the control of an MSG job. There are two types of MSG jobs. Each MSG configuration includes a single, "central" MSG job responsible for system initialization and interhost communication. In addition, there may be one or more "process-controlling" MSG jobs responsible for directly controlling communicating processes. The process-controlling MSG jobs interact with one another and the central MSG as necessary to support process communication.

To run an MSG configuration, a central MSG job and one or more process-controlling MSG jobs must be created. This initialization may be accomplished either manually or automatically.

3.1.1. Manual Startup.

MSG may be initialized manually by starting a central MSG job and zero, one or more process-controlling jobs. The following EXEC command sequence can be used to start a central MSG job:

```
@GET MSG.SAV
@REENTER
```

or

```
@GET MSG.SAV
@DETACH (INFILE) - (OUTFILE) - (AND) START
```

In the first case, the central MSG job will print initialization information including the list of known generic names and remote host MSG contact sockets. In the second case, no such information will be displayed since the central MSG will start up detached. The known generic names and remote host MSG contact sockets are defined by configuration control files (see Section 3.1.4).

The central MSG will, depending upon internal switch settings (see Section 3.1.3) and the configuration control files, start up various process-controlling MSG jobs.

Additional process-controlling MSG jobs can be started by logging in, issuing the following EXEC command:

@RUN MSG.SAV

and interacting with MSG to specify the process class to be supported. After the process class has been specified, the user is given the opportunity to have MSG create and start a process. If the user chooses to have a process started, he is also given the opportunity of specifying a debugger (DDT, IDDT, or BDDT) for the process. If he chooses not to have a process started, then no processes will execute under control of the job until the job allocates one to receive a generically addressed message or the user later explicitly starts a processvia the START command (see Section 3.2).

3.1.2. Automatic Startup.

MSG can be started automatically whenever TENEX is initialized by adding MSG.SAV to the system autojob startup file. If this is done, when TENEX is restarted, a (detached) central MSG job will be created and started automatically. As with manual startup, the central MSG job will, depending upon internal switch settings and the configuration control files, start up various process-controlling MSG jobs.

3.1.3. Internal MSG Settings.

a. MSG uses a number of files in its initialization and as part of its normal operation. It expects these files to be found in a particular, user-specified directory. When it is running, MSG is connected to that directory. The MSG directory can be defined by the following sequence of commands:

```
@GET MSG.SAV
@DDT

MKMSG$G
 Directory for MSG: XYZ
@SSAVE (PAGES FROM) Ø (TO) 677 (ON) MSG.SAV
```

b. An internal switch, called DBGSw, controls several aspects of MSG operation. The switch settings are as follows:

Bit Ø: on: Use directory relative (see (a) above) sockets for local MSG contact socket.
off: Use absolute socket (29.) for local MSG contact socket (requires TENEX absolute socket capability).

Bit 1: on: Top fork of central MSG job should halt on all inferior fork terminations (useful for debugging MSG).

Bit 2: on: Don't create local inter-MSG shared data base (useful for debugging MSG).

Bit 3: on: Don't create any process-controlling MSG jobs as part of MSG initialization. (This overrides any declarations in the generic name definition file).

Bit 4: on: Do "forced" SETNM to allow TENEX to accumulate statistics on MSG operation.

The folllowing settings are recommended for debugging interacting processes:

DBGSW: 1B0+1B3

3.1.4. Configuration Control.

There are two text files used to define MSG configurations: the generic name file and the network configuration file. Both files are expected to be in the MSG directory (see 3.1.3(a) above).

The generic name file serves to define the generic process classes known to MSG. It is named:

(SP)(SP)MSG-GENERIC-NAMES.;

When the central MSG is started, it initializes an internal generic name table by reading the generic name file. In addition, new names may be added dynamically to the internal generic name table by running a process-controlling MSG.

The generic name file is a text file. It consists of a list of name definitions, each of which is a line of the form:

Name Code Create-Spec Terminate-Spec Saved-File-Name

or:

Name Code Create-Spec Host-Spec.

The first form is used to define locally supported processes: "Name" is the generic class being defined; "Code" is the MSG-to-MSG internal code for the generic class (an integer <128.); "Create-Spec" specifies how generically addressed messages or the class are to be handled by MSG when there are no outstanding ReceiveGenericMessage primitives by existing processes in the class; "Terminate-Spec" defines how the StopMe primitive is to be handled when a process in the class executes it; "Saved-File-Name" is the name of the saved file for the process core image.

The second form is used to define remotely implemented processes. Here "Create-Spec" must be the string "REMOTEJOB"; "Host-Spec" is either the ARPANET name (a text string) or the ARPANET address (an octal integer) for the host which supports the process class.

Create-Spec is an expression of the form:

CSpec

or

CSpec,CModifier.

At present the following six CSpecs are defined:

CLASSJOB - All processes in the class are to execute in a TENEX job dedicated to that class.

INITCLASSJOB - Same as CLASSJOB with the exception that when MSG is started a null generic message will be sent to a process in the class.

MISCJOB - All processes in the class are to execute in a TENEX job dedicated to "miscellaneous" processes.

NEWJOB - Each process in the class is to execute in a TENEX job by itself.

REMOTEJOB - Processes in this class execute on some other host.

SINGLEPROC - At most, only a single process of this type may exist at any time.

CLASSJOB and REMOTEJOB are the most meaningful CSpecs when operating in a debugging environment.

The CModifier is used to control the action taken by the central MSG with respect to a given generic process class at initialization time. The following two CModifiers are defined:

JOB - MSG is to create and start a process-controlling MSG job for the class at initialization time. This is the default CModifier if none is specified.

NOJOB - MSG should not create a process-controlling job for the class at initialization time.

The NOJOB modifier would be an appropriate CModifier for NSW Front End (FE) processes which are started up by the NSW dispatcher. In addition, it should be useful in a debugging environment where it is desirable to have a process class defined at initialization by the central MSG but run under programmer and debugger control via a manually started process-controlling MSG.

At present the following three Terminate-Specs are defined:

KILLPROC - When the process executes StopMe, kill it (via the TENEX KFORK JSYS).

RESTARTPROC - When the process executes StopMe, assign a new MSG process name to the TENEX fork(s) which implements it and restart the fork at its start address.

STOPMSG - Same as KILLPROC with the exception that if no more processes exist in the job, the process-controlling MSG will terminate.

When a new process class is defined by running a process-controlling MSG, CLASSJOB and KILLPROC are used as the Create-Spec and Terminate-Spec, respectively.

The MSG network configuration file serves to specify how a local MSG should contact other MSGs running on remote hosts. The name of the file is:

(SP)(SP)MSG-NETWORK-CONFIGURATION.;

The MSG configuration file is a text file. It consists of a list of host/ICP-contact-socket specifications, each of which is a line of the form:

Host-Spec Socket-Spec

where Host-Spec is either the ARPANET name (a text string) or the ARPANET address (an octal integer) for a remote host; and, Socket-Spec is the MSG ICP contact socket at that host (an odd octal integer =< 37777777777).

As part of its initialization procedure, the central MSG looks for an MSG network configuration file. If one is found, it is used to initialize host tables internal to MSG. The ICP contact sockets for hosts not specified in the file are assumed to be 35 (octal). If no configuration file is found and if the central MSG job has a controlling terminal, then the user will be asked to supply host/socket pairs. As before, the contact socket for any hosts not specified will be assumed to be 35 (octal). If there is no file and if MSG is running as a detached job, then socket 35 (octal) will be assumed to be the MSG contact socket for all remote hosts.

3.1.5. MSG ERRHLTs.

An error halting (ERRHLT) mechanism is included in MSG to facilitate debugging of MSG. As part of its normal operation MSG performs a variety of internal consistency checks. If an MSG job detects an inconsistency, it executes an ERRHLT procedure. As part of this procedure it prints a message which indicates the nature of the inconsistency and where it was detected. Persistent ERRHLTs should be reported to the MSG implementers.

3.1.6. Abnormal Process Termination.

MSG processes normally terminate by executing the StopMe primitive. Other forms of termination (e.g., execution of the HALTF JSYS, forced termination due to an illegal instruction or i/o data error, etc.) are regarded by MSG as abnormal. In such a case, MSG will report the abnormal termination by printing the process name, the process PC when the termination occurred, and the type of termination. If a debugger for the process has been specified (see Sections 3.1.1 and 3.2), MSG will pass control to the debugger for the process. Otherwise, MSG will take no further action with respect to the process until the user either TERMINATEs the process or attempts to debug it (via the DEBUG command described in Section 3.2).

3.1.7. Terminating MSG Jobs.

The MSG jobs comprising a TENEX MSG configuraton interact with one another via a shared data base. It is critical to the operation of the MSG configuration that this data base always be in a consistent state. To insure that this is the case, the various jobs use a locking discipline when they modify certain portions of the data base.

It is important that termination of an MSG job is done in an orderly way that preserves the consistency of the shared data base, if the remainder of the MSG configuration is to continue running properly. In particular, it is not safe to type CONTROL-C and RESET (or LOGOUT) to terminate an MSG job (unless the entire configuration is to be terminated). Doing so will cause internal resources allocated to the job to remain allocated to the (now) non-existant job and, in addition, it may leave portions of the shared data base locked.

The proper way to terminate an MSG job is to use the QUIT or RESTART commands (See Section 3.2).

At certain critical points of its operation MSG disables most normal interrupts in order to protect the shared data base. For example, CONTROL-C is disabled. If it is necessary to stop an MSG job that is not responsive to CONTROL-C (for example, because it is suspected that either the MSG job or a process it is managing is looping), the user should type CONTROL-P, the MSG "panic" interrupt". CONTROL-P stops an MSG job in a way that will not permit the job to be restarted. For this reason, CONTROL-P should be used only in panic situations.

3.2. Monitoring and Controlling MSG Processes.

The TENEX MSG includes facilities that allow a user to monitor and control process activity. These facilities are accessible through an MSG command language interpreter which can be activated by typing CONTROL-S to a running MSG job. When the command language interpreter is activated, all other activity in the MSG job is suspended until the user either explicitly or implicitly deactivates the command language interpreter.

The prompt character for the MSG command language interpreter is ">". Several of the standard TENEX line editing characters are available: CONTROL-A (character erase), CONTROL-R (retype line), ESCAPE (complete field and prompt for parameter), and RUBOUT (abort command line).

A list of the available commands together with a brief description of each can be obtained by typing "?<cr>". This list is printed below followed by further discussion:

ALL (Jobs)
: Prints summary information on all active MSG jobs.

CONTINUE
: Deactivates the command language interpreter and resumes normal operation of the MSG job.

DEBUG (Process) nnn
: Invokes a debugger for the specified MSG process. Takes numeric argument of PCB # of process.

FORCE (Timeout of PEs) nnn
: Forces timeout of a specific pending event (nnn) or all pending events (if nnn is not specified).

GENERIC
: Prints information about defined Generic classes.

HOSTS
: Prints information about hosts known to MSG.

JOB (Status)
: Prints detailed information on local MSG job. Optionally takes numeric argument of JCB # of another MSG job.

LOGGING (On or Off)
: Turns binary event logging on or off. To get text form of log use UPDATE command.

MINIMUM (Timeout as HH:MM:SS) hh:mm:ss
Prints current minimum pending event timeout and resets it to specified value if one is given (max of 24:00:00).

MSGDDT
Invokes DDT for debugging MSG.

PROCESS (Status) nnn
Prints detailed information on specified MSG process. Takes numeric argument of PCB # of process.

QUIT
Tries to clean up local MSG job and then halts.

RECENT (User Primitives)
Prints information on recently issued user primitive requests. Takes optional numeric argument of max number of these to give.

RESTART (MSG)
Tries to clean up local MSG job and then restart it.

START (Process)
Creates and starts a process.

TERMINATE (Process) nnn
Terminates the specified process. Takes numeric argument of PCB # of process.

UPDATE (Text Log File)
Updates text form of event log. Converts any log entries inserted since last UPDATE command. The name of the text log file is MSG-LOG.TXT. Its version number will be the incarnation number of the running MSG.

The status information printed for an MSG job includes its Job Control Block (JCB) number (which is used internally by MSG), its TENEX job number, the type of MSG job it is: either Control (=central); Local (=process-controlling); or Not In Use. The job print out will flag any job that has ERRHLTed (see Section 3.1.5). The JOB command also prints the following: any processes managed by the job, any completed pending events that have not yet been delivered to processes, any pending generic message control blocks (MCBs) for which a process has not yet been allocated, and any connection control blocks (CCBs) for direct connections that processes in the MSG job have created.

The status information printed for a process includes its Process Control Block (PCB) number (which is used internally by MSG), its MSG Process Name, its process state (normally Active or Stopped), and the TENEX fork handle for the process. If the process is controlled by the local job, then its TENEX fork status and PC are also printed. Any pending events associated with a process are also printed.

Whenever a process executes an MSG primitive an entry is made in a circular buffer (the Recent User Primitive table). This entry contains information about the call including any associated pending event (PE). The table (in the current implementation) holds 20 entries. A print out of recently executed user primitives can be obtained by the RECENT command which prints information about the requested number of primitives (up to 20) in reverse chronological order. The information printed for each primitive includes the time it was executed, the type of primitive, its argument (contents of AC1), whether the primitive was valid (if not valid the error code is given), and the MSG name for the executing process. If a pending event was created (or referenced, as by Rescind) then the PE is also given.

Note that as long as there is a PE generated by a process in the Recent User Primitive table the process' PCB will not be released even if the process has executed a StopMe or has been otherwise terminated. Similarly, the JCB of the MSG job that created the PCB will not be released. Thus "Not In Use" JCBs and "Stopped" PCBs will be shown by the status reporting commands until all PE's associated with them in the Recent User Primitive table have been released.

The information printed for a pending event includes the PE number, the type of PE, the addressee (if any), the state of the PE (usually Pending, Succeeded, Timed Out, or a numeric Error code), and the type of signal requested by the user process.

Where appropriate, information is printed for Alarm, Message, or Connection Control Blocks (ACBs, MCBs, or CCBs). The information printed includes the names of sending and receiving processes, the local state of the control block (Pending, Timed Out, Aborted, or Completed), the remote state of the control block (usually Pending or Complete; this may also include the state of the transaction relative to the MSG-to-MSG protocol), Alarm Code for ACBs, and further connection information for CCBs.

MSG can be instructed to log events of interest in a permanent log file via the LOGGING command. This feature is useful in debugging or performance monitoring situations when more than the last 20 primitives are of interest. The events logged include execution of primitives, creation of MSG jobs, and creation of processes. Because event logging slows MSG somewhat, the default is LOGGING OFF.

When logging is enabled, MSG maintains an event log in a file which has a binary format. The UPDATE command can be used to generate a textual version of the log. When the UPDATE command is executed, any entries made to the binary log since the last UPDATE command are converted to text form and appended to the textual log file.

The START command may be used to create and start a new MSG process. Before the process is started, the user has the opportunity to specify a debugger for the process (DDT, IDDT or BDDT). If a debugger is specified, control is passed to it from which the user may start the process. In either case, the START command "completes" by deactivating the command language interpreter and resuming normal MSG operation by the job.

The DEBUG command invokes the debugger DDT for a specified process. Control is passed to DDT after process status information (including contents of its ACs, the process PC and its execution state) is printed. The DEBUG command "completes" by resuming normal MSG operation of the job.

The CONTINUE command deactivates the command language interpreter and causes normal MSG operation by the job to resume.

The QUIT command terminates the MSG job in an orderly way (see Section 3.1.7). It does this by terminating all processes controlled by the job, aborting any pending events associated with the processes, and deallocating any other resources it may be using.

The RESTART command may be used to terminate and restart an MSG job. Its effect is equivalent to QUIT followed by RUN MSG.SAV (see section 3.1.1).

APPENDIX

MSG Disposition Error Codes.

The following are 16 bit error codes and are the codes that appear in the disposition field of the parameter block associated with various primitive call:

100001	Signal given is invalid.
100002	Handling given is invalid.
100003	Process name given is invalid.
100004	Unable to map up parameter block.
100005	Primitive not implemented.
100006	Invalid host address in process name.
100007	Unable to map parameter block down.
100010	Can't convert from internal proc name to str.
100101	Unable to map message page in Message Send.
100102	Message length invalid.
100103	Generic code malfunction.
100201	Alarm Accept code not 0 or -1.
100301	Enable Alarm already outstanding.
100401	Invalid Event Handle in Rescind primitive.
100402	Unable to Rescind.
100601	Already have connection of that ID.
100602	Unknown connection.
100603	Connection not to process named.
100604	Remote site refused connection.
140004	MSG message too long.
140101	Destination process unknown.
140102	Destination process message queue full.
140103	Destination name/handling - generic/specific mismatch.
140104	Generic name not legal for destination process.
140105	Bad incarnation number on destination process.
140301	Insufficient resources to complete command.
140401	Process not accepting alarms now.
140402	Alarm already queued for process.
140501	That generic class not supported here.
140502	Can't allocate a process for generic message.
140601	User process closed connection.
140603	Received a CLS for connection.
140604	Invalid connection type.
140605	Remote process name mis-match.
140606	Referenced connection transaction does not exist.
140611	Connection type mis-match.
140612	Connection aborted.

Other values for the disposition field include:

Ø	Success disposition.
2	Timed out.
3	Error in delivery of message.
4	Must resynch message stream.
5	Primitive aborted by user process.